

SPECIFICATION

Electronic Version 1.2.8

Stylesheet Version 1.0

[TRANSMISSION CONVERGENCE SUBLAYER CIRCUIT AND OPERATING METHOD FOR ASYNCHRONOUS RECEIVER]

Cross Reference to Related Applications

This application claims the priority benefit of Taiwan application serial no. 90122431, filed September 11, 2001.

Background of Invention

[0001] Field of Invention

[0002] The present invention relates to the circuit and operating method of an asynchronous transmission receiver. More particularly, the present invention relates to the transmission convergence sublayer circuit and operating method of an asynchronous transmission receiver.

[0003] Description of Related Art

[0004] In a communication system, data is transferred from an emission system to a reception system through a transmission medium. The transmitted data is assembled together according to specified communication protocols in several layers. Similarly, the reception system processes the received data layer by layer according to the communication protocols. Among the communication protocols, the lowest layer unit is the physical layer. In asynchronous transfer mode, the physical layer is further divided into two units, namely, a physical medium and a transmission convergence sublayer.

[0005] Fig. 1 shows the data format of cell data processed by the transmission convergence sublayer at the transmission terminal working in the asynchronous transfer mode. The data format is decided by international telecommunication union (ITU) according to broadband integrated service digital network (B-ISDN) proposal ITU-T I.432. The proposed specification stipulates that the data format of a cell using an asynchronous transfer mode must have a size capable of accommodating 53 bytes. The 53 bytes include 5 bytes of header and 48 bytes of the so-called payload. The 5 bytes of header further comprises 4 bits of general flow control (GFC) code, 8 bits of virtual transmission path identification (VPI) code, 16 bits of virtual transmission channel identification (VCI), 3 bits of package type (PT) code, 1 bit of loss package classification (CLP) code and 8 bits of header error control (HEC) code.

[0006] To ensure correctness of header cell data at the receiving terminal, the first 32 bits in the header cell is applied to a cyclic redundancy check (CRC) polynomial $X^8 + X^2 + X + 1$ to produce an 8-bit header cyclic redundancy code. Fig. 2A is a block showing a conventional method of using a header cyclic redundancy code generator at the emission terminal to produce header cyclic redundancy code. The circuit in Fig. 2A is capable of generating necessary header cyclic redundancy code for data error detection.

[0007] The upper layer unit generates transmission data according to an asynchronous transfer communication protocol. The transmission data at the transfer terminal of the transmission convergence sublayer is scrambled to produce the payload within the data cell according to a scrambling polynomial $X^{43} + 1$.

[0008] In general, the asynchronous transfer mode is structured upon a synchronized transmission system with a fixed bandwidth. When nothing is transmitted from an upper layer unit, the transmission convergence sublayer must generate an idle cell having special header and payload and the idle cell data must be transferred to a physical medium for transmission rate matching. The processing work required to be performed by the transmission convergence sublayer at the receiving terminal in an asynchronous transfer mode includes receiving a data stream and comparing the data stream with header cyclic redundancy code to find the header cell. Ultimately, data cells are correctly positioned and synchronously received. Once such

synchronized state is reached, correctness of the header cell data in subsequently received data cells are checked and the payload within the data cell is descrambled. If the header cell is found to be correct and the data cell is not an idle cell, the header cyclic redundancy code in the header cell is removed. The data cell is rearranged to form a word and the word is written into a buffer. Finally, the word data is transferred to an upper layer for subsequent treatment.

[0009] Fig. 2B is a block diagram showing a conventional data cell synchronizing circuit for a receiver terminal operating in an asynchronous transfer mode. Reference is made to Fig. 2B for synchronizing reception of data cells in the transmission convergence sublayer by an asynchronous receiving terminal and the method of checking the correctness of header cells within the data cells received after synchrony.

[0010] In Fig. 2B, a modulo 2 adder 202, a D-type flip-flop 204 and a cyclic redundancy check arithmetic operation circuit 206 together form a long division circuit. The number to be divided is the first 40 bits of data in the data cell and the divisor is the polynomial $X^8 + X^2 + X + 1$. If the result of calculation is correct, a decoder 208 decodes the computed value to obtain a cell synchronizing pulse. On the other hand, if the result of calculation is incorrect, 8 bits of data move in from the data cell to conduct a division. However, the earliest 8 bits of the previously divided 40 bit data must be corrected to eliminate any effect in the next round of division operation. The circuit comprising another modulo 2 adder 210, D-type flip-flop 214 and remainder arithmetic operation circuit 212 serve to eliminate the effect the 8 bit data has on the next round of division operation.

[0011] The function of the circuit in Fig. 2B is to operate on the received data stream. Through a comparison with the header cyclic redundancy code, the header cell is found. Hence, the data cells are received in synchrony. Furthermore, after data cell synchronization, header cyclic redundancy code comparison of subsequently received data cells continues.

[0012] However, to descramble the payload within a data cell, rearrange the data format from byte groups to word groups or double word groups and submit to buffer for processing by the upper layer unit, additional secondary circuit stages must be introduced. Hence, synchronized reception, header inspection, data descrambling and

data format rearrangement must rely on the complicated integration of circuits such as the one shown in Fig. 2B and any additional secondary circuit stages. Consequently, data processing takes longer to complete.

Summary of Invention

[0013] Accordingly, one object of the present invention is to provide a transmission convergence sublayer circuit and operating method for receiving an asynchronous transmission. The circuit has an optimized structure capable of synchronizing data reception and conducting header inspection, data descrambling as well as data format rearrangement in the shortest possible time.

[0014]

To achieve these and other advantages and in accordance with the purpose of the invention, as embodied and broadly described herein, the invention provides a transmission convergence sublayer circuit. When the data stream enable signal terminal of a deframer is enabled, the transmission convergence sublayer circuit receives data cells from the deframer. The data cells comprise of a number of byte groups including a header and a payload. The transmission convergence sublayer circuit includes a byte-wise data pipeline, a header cyclic redundancy checker, an idle cell identifier, a cell delineation state machine, a byte pointer, a descrambler and a write-in buffer controller. The byte-wise data pipeline sequentially picks up and holds byte data temporarily. The header cyclic redundancy checker receives the byte data and issues a syndrome code to indicate whether a header appears in the data. The idle cell identifier determines if the data received by the byte-wise data pipeline include a non-idle data cell. The cell delineation state machine determines the transmission state of the data cells according to the content and frequency of the syndrome code and submits a state signal to indicate the current state. The state signal distinguishes between a searching state and a complete synchronization state. The byte pointer provides sequence labels to those bytes belonging to the data cell newly picked up by the byte-wise data pipeline. Since the newly received bytes are subsequently transformed into double word groups, the sequence labels serve as address pointers for transferring the double word groups to a buffer. The descrambler descrambles the byte groups temporarily hold up in the byte-wise data pipeline and transfers the results to the buffer. The write-in buffer controller writes the descrambled data into

the buffer according to the indication provided by the byte pointer when the idle cell identifier and the cell delineation state machine grant the permission to do so.

[0015] This invention also provides an operating method of the transmission convergence sublayer of an asynchronous transmission receiver for receiving data cells from a deframer and data stream enable signal. The data cell comprises multiple-byte groups belonging either to a header or a payload. The operating method includes the following steps. First, a byte-wise data pipeline receives bytes of data. The byte-wise data pipeline sequentially picks up data bytes and stores a specified number of bytes temporarily. A header cyclic redundancy checker receives the data bytes synchronously, determines if a header is present in the data and submits a syndrome code that indicates the presence or absence of a header. According to the syndrome code, a cell delineation state machine determines if a state transition from a searching state to a completely synchronized state is carried out. If the cell delineation state machine effects a state transition to the completely synchronized state, a descrambler descrambles data bytes in the byte-wise data pipeline that have a capacity to decode a double word group. A byte pointer outputs a pointer signal according to the state indicated by the cell delineation state machine. The pointer signal indicates the sequence number of the newly received bytes within the data cell and the address for storing data decoded by the descrambler.

[0016] In brief, this invention provides an optimized circuit design for synchronizing reception of data and conducting header inspection, data descrambling as well as data format rearrangement in the shortest possible time.

[0017] It is to be understood that both the foregoing general description and the following detailed description are exemplary, and are intended to provide further explanation of the invention as claimed.

Brief Description of Drawings

[0018] The accompanying drawings are included to provide a further understanding of the invention, and are incorporated in and constitute a part of this specification. The drawings illustrate embodiments of the invention and, together with the description, serve to explain the principles of the invention. In the drawings,

- [0019] Fig. 1 shows the data format of cell data processed by the transmission convergence sublayer at the transmission terminal working in the asynchronous transfer mode;
- [0020] Fig. 2A is a block diagram showing a conventional method of using a header cyclic redundancy code generator at the emission terminal to produce header cyclic redundancy code;
- [0021] Fig. 2B is a block diagram showing a conventional data cell synchronizing circuit for a receiver terminal operating in an asynchronous transfer mode;
- [0022] Fig. 3 is a block diagram showing a conventional system comprising a buffer, a transmission convergence sublayer and a deframer;
- [0023] Fig. 4 is a block diagram showing a transmission convergence sublayer system for receiving asynchronous transfer according to a first embodiment of this invention;
- [0024] Fig. 5A is a diagram showing all the possible states of the cell delineation state machine according to this invention;
- [0025] Fig. 5B is a block diagram showing a circuit implementation of the cell delineation state machine according to this invention;
- [0026] Fig. 6 is a diagram showing a circuit implementation of the cell counter according to one preferred embodiment of this invention;
- [0027] Fig. 7A is a diagram showing the states after the rearrangement of bytes within the data cells into byte addresses and byte pointers;
- [0028] Fig. 7B is a diagram showing a circuit implementation of the byte pointer according to one preferred embodiment of this invention;
- [0029] Fig. 8A is a diagram showing a circuit implementation of the header cyclic redundancy checker according to one preferred embodiment of this invention;
- [0030] Fig. 8B is a diagram showing a circuit implementation of the remainder compensation unit according to one preferred embodiment of this invention;
- [0031] Fig. 8C is a diagram showing a circuit implementation of the quotient feedback

unit according to one preferred embodiment of this invention;

[0032] Fig. 8D is a diagram showing a circuit implementation of the header cyclic redundancy checker according to another preferred embodiment of this invention;

[0033] Fig. 8E is a diagram showing a circuit implementation of the header cyclic redundancy checker according to yet another preferred embodiment of this invention;

[0034] Fig. 9 is a diagram showing a circuit implementation of the byte-wise data pipeline according to one preferred embodiment of this invention;

[0035] Fig. 10 is a diagram showing a circuit implementation of the descrambler according to one preferred embodiment of this invention;

[0036] Fig. 11A is a diagram showing a data format of the idle cell according to this invention;

[0037] Fig. 11B is a diagram showing a circuit implementation of the idle cell identifier according to one preferred embodiment of this invention;

[0038] Fig. 12 is a diagram showing a circuit implementation of the write-in buffer controller according to one preferred embodiment of this invention;

[0039] Fig. 13 is a block diagram showing a transmission convergence sublayer system for receiving asynchronous transfer according to a second embodiment of this invention;

[0040] Fig. 14 is a lookup reference table for modifying bit errors; and

[0041] Fig. 15 is a diagram showing a circuit implementation of the descrambler for connecting with the header bit error corrector according to one preferred embodiment of this invention.

Detailed Description

[0042] Reference will now be made in detail to the present preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. Wherever possible, the same reference numbers are used in the drawings and the description to refer to the same or like parts.

receiving asynchronous transfer according to a first embodiment of this invention. As shown in Fig. 4, a cell delineation state machine inside the asynchronous transfer mode transmission convergence sublayer 302 receives a data stream enable signal AtmRx_Hit from a deframer (not shown), a counter signal from a cell counter 406, a syndrome code from a header cyclic redundancy checker 410 and an pointer signal from a byte pointer 404. The data stream enable signal AtmRx_Hit controls the operation of the cell delineation state machine 402. According to the counter signal, the syndrome code and the pointer signal, the cell delineation state machine 402 decides if the current state needs to be changed and transmits a plurality of state signals to other devices.

[0045] Fig. 5A is a diagram showing all the possible states of the cell delineation state machine according to this invention. As shown in Fig. 5A, the cell delineation state machine 402 includes a plurality of states for determining the work that currently needs to be performed by the asynchronous transfer mode transmission convergence sublayer 302.

[0046] In the header search state, the asynchronous transfer mode transmission convergence sublayer 302 receives consecutive data cells from the deframer (not shown) and the header cyclic redundancy checker 402 carries out a computation of the header of the data cells to find an 8-bit syndrome code. If the cell delineation state machine 402 finds the syndrome code is not 0x00, the cell delineation state machine 402 maintains its current state. On the other hand, if the cell delineation state machine 402 finds the syndrome code to be 0x00, the cell delineation state machine 402 proceeds to a pre-synchronization state.

[0047] In the pre-synchronization state, the asynchronous transfer mode transmission convergence sublayer 302 continues to receive consecutive data cells from the deframer (not shown) and the header cyclic redundancy checker 410 continues to carry out header computation to find the syndrome code. If the cell delineation state machine 402 finds the syndrome code within the consecutive data cells is not 0x00, the asynchronous transfer mode transmission convergence sublayer 302 returns to the header search state. On the other hand, if the cell delineation state machine 402 finds the syndrome code to be 0x00, the asynchronous transfer mode transmission

convergence sublayer 302 maintains the original state and increments the pre-synchronization state counter. When the counter reaches a value DELTA after receiving the consecutive data cells, the asynchronous transfer mode transmission convergence sublayer 302 proceeds into a full synchronization state.

[0048] In the full synchronization state, the asynchronous transfer mode transmission convergence sublayer 302 continues to receive consecutive data cells from the deframer (not shown) and the header cyclic redundancy checker 410 continues to carry out header computations to find the syndrome code. If the syndrome code is 0x00, the cell delineation state machine 402 maintains the original state and removes the header cyclic redundancy code within the header utilizing the byte pointer 404 and the byte-wise data pipeline 408. The payload within the data cell is extracted and descrambled by the descrambler 412. The word group format is transformed into double word format. The transformed double word format data is transferred to the buffer (not shown). On the other hand, if the cell delineation state machine 402 finds that the syndrome code computed from header computation of the consecutively received data cells is not 0x00, the asynchronous transfer mode transmission convergence sublayer 302 proceeds into a synchronization conservation state.

[0049] In the synchronization conservation state, the asynchronous transfer mode transmission convergence sublayer 302 continues to receive consecutive data cells from the deframer (not shown) and the header cyclic redundancy checker 410 continues to carry out header computation to obtain the syndrome code. If the cell delineation state machine 402 finds the syndrome code is 0x00, the asynchronous transfer mode transmission convergence sublayer 302 returns to the full synchronization state. On the other hand, if the syndrome code is not 0x00, the asynchronous transfer mode transmission convergence sublayer 302 maintains its original state and increments the synchronization conservation state counter. Within the synchronization conservation state, when the number of non-specialized syndrome code of the received data cell accumulates to a value ALPHA-1, the asynchronous transfer mode transmission convergence sublayer 302 proceeds into the header search state.

[0050]

Fig. 5B is a block diagram showing a circuit implementation of the cell delineation

state machine according to this invention. As shown in Fig. 5B, under various states, a secondary state estimator 502 inside the cell delineation state machine 402 performs an estimation to produce a secondary state code N_State according to the current state code C_State from a D-type flip-flop 504, the counter signal Cnt_Max from the cell counter 406 (refer to Fig. 4), the syndrome code from the header cyclic redundancy checker 410 (refer to Fig. 4) and the pointer signal Ptr_03 from the byte pointer 404 (refer to Fig. 4). For example in Fig. 5A, assume the current state is pre-synchronization state having a state code of 0x02. When the pointer signal Ptr_03 is "1" (a high potential) and the number of times the syndrome code computed from the header of consecutively received data cells is 0x00 accumulates to the value DELTA, the cell counter 406 issues a count signal Cnt_Max "1" so that the secondary state code N_State becomes 0x04 (that is, a full synchronization state).

[0051] The deframer 306 (refer to Fig. 3) issues a data stream enable signal AtmRx_Hit to enable the D-type flip-flop 504. In a synchronizing pulse cycle, the secondary state code N_State fed to the input terminal D of the D-type flip-flop 504 serves as the current state code C_State at the output terminal Q of the D-type flip-flop 504.

[0052] A decoder 506 receives the current state code C_State from the output terminal of the D-type flip-flop 504 and the secondary state code N_State from the secondary state estimator 502 and decodes them into a plurality of current state signals (such as CS_Hunting, CS_Presync, CS_Corsync and CS_Detsync as shown in Fig. 5B) and a plurality of secondary state signals (such as NS_Hunting, NS_Presync, NS_Corsync and NS_Detsync as shown in Fig. 5B). If the state is pre-synchronization state as aforesaid and the secondary state is full synchronization state, the current state signal CS_Presync and the secondary state signal NS_Corsync are at logic "1" potential while the rest of the signals are at logic "0" potential (a low level).

[0053] In Fig. 4, the cell counter 406 is coupled to the cell delineation state machine 402 and the byte pointer 404. The deframer 306 (refer to Fig. 3) transmits the data stream enable signal AtmRx_Hit to enable the cell counter 406. The cell counter 406 receives the pointer signal Ptr_03 from the byte pointer 404 and the state signal CS_Hunting and CS_Corsync from the cell delineation state machine 402. The cell counter 406 also determines if the cumulative counter value reaches the pre-defined DELTA or ALPHA-

1 value according to the state signal CS_Presync. Once the counter counts up to the pre-defined values, a count signal Cnt_Max is sent to the cell delineation state machine 402.

[0054] Fig. 6 is a diagram showing a circuit implementation of the cell counter according to one preferred embodiment of this invention. As shown in Fig. 6, a first input terminal of an AND gate 602 within the cell counter 406 receives the pointer signal Ptr_03 from the byte pointer 404 (refer to Fig. 4), a second input terminal of the AND gate 602 receives the data stream enable signal AtmRx_Hit from the deframer 306 (refer to Fig. 3), and the output terminal of the AND gate 602 submits an ANDed signal to the ENA terminal of a counter 606. A first input terminal of a NOR gate 604 receives the signal CS_Hunting from the cell delineation state machine 402 (refer to Fig. 4). A second input terminal of the NOR gate 604 receives the signal CS_Corsync from the cell delineation state machine 402 (refer to Fig. 4). The output terminal of the NOR gate 604 submits a NORed signal to the CLRN terminal of the counter 606. When the cell delineation state machine 402 (refer to Fig. 4) is in the header search state or a full synchronization state, the NOR gate 604 prevents the counter 606 from counting.

[0055] When the clock terminal CLK is operating within a synchronizing pulse cycle, if the ANDed signal sent from the AND gate 602 to the ENA terminal is a logic "1", the 4-bit count value Cell_Cnt at the output terminal Q of the counter 606 will increment by one automatically. If the NORed signal sent from the NOR gate 604 to the CLRN terminal is a logic "0", the 4-bit count value Cell_Cnt at the output terminal Q of the counter 606 will be cleared to zeros.

[0056] A first input terminal of a multiplexer 608 within the cell counter 406 receives a DELTA value and a second input terminal of the multiplexer 608 receives an ALPHA-1 value. A selection terminal of the multiplexer 608 picks up the state signal CS_Presync from the cell delineation state machine 402 (refer to Fig. 4) to determine if the output terminal of the multiplexer 608 outputs the DELTA value (when the state signal CS_Presync is at logic "1") or outputs the ALPHA-1 value (when the state signal CS_Presync is at logic "0").

[0057] A first input terminal of a comparator 610 within the cell counter 406 receives the

count value Cell_Cnt and a second input terminal of the comparator 610 receives the DELTA value or the ALPHA-1 value. If the count value Cell_Cnt is equal to the DELTA value or the ALPHA-1 value, the output terminal of the comparator 610 outputs a count signal Cnt_Max to the cell delineation state machine 402 (refer to Fig. 4).

[0058] In Fig. 4, the byte pointer 404 is coupled to the cell delineation state machine. Hence, the byte pointer 404 is capable of receiving a data stream signal AtmRx_Hit from the deframer 306 (refer to Fig. 3) so that the byte pointer 404 is enabled to produce the byte address corresponding to the count value of the current data cell. Fig. 7A is a diagram showing the states after the rearrangement of bytes within the data cell into byte addresses and byte pointers. A complete data cell has 53 bytes. Byte index is labeled from 0 ~ 52 and byte pointer is labeled from 0 ~ 51 because the byte indexes labeled 4 and 5 are assimilated together and hence having the same byte pointer label 4. With this arrangement, sequence number of the bytes belonging to the newly input data cells transmitted into the byte-wise data pipeline 408 is set up. In addition, the highest 4 bits of the pointer signal output from the byte pointer 404 can be directly retrieved to serve as an indicator showing the sequence number of each batch of double word data within the data cells. Ultimately, address for the batch of double word data can be clearly indicated. Details are further described below.

[0059] By repeating the pointer positions of byte 4 and 5, the remaining 52 pointed positions is labeled from 0 to 51. Hence, only 6 bits is necessary to represent the 52 bytes. Therefore, after receiving the 8th byte or in other words the first batch of double byte data, the pointer signal will point to the seventh batch, that is, 000111. Taking the highest four bits, a pointer position 0001 that represents the address pointer of the first batch of double word data is obtained. The reason for the possibility of such an arrangement is that the header cyclic redundancy codes are specially added codes for determining if there is any data transmission error during a transmission session rather than data with real applications. Hence, the codes may be removed without any effect on real applications.

[0060] Fig. 7B is a diagram showing a circuit implementation of the byte pointer according to one preferred embodiment of this invention. As shown in Fig. 7B, a first input terminal of an OR gate 702 within the byte pointer 404 receives a header search

state signal NS_Hunting from the cell delineation state machine 402 (refer to Fig. 4). A second input terminal of the OR gate 702 receives the pointer signal Ptr_03 from a decoder 712. The output terminal of the OR gate 702 outputs an ORed signal OR1.

[0061] An input terminal D of a D-type flip-flop 704 within the byte pointer 404 receives the signal OR1. The synchronizing pulse received by the clock terminal CLK and the data cell enable signal AtmRx_Hit received by the ENA terminal of the D-type flip-flop 704 enable the D-type flip-flop to output a mask signal at the output terminal Q.

[0062] The "0" input terminal of a multiplexer (MUX) 714 within the byte pointer 404 receives the value 0x00 while the "1" input terminal of the multiplexer 714 receives the value 0x03. The select terminal of multiplexer 714 receives a secondary header search state signal NS_Hunting from the cell delineation state machine 402 (refer to Fig. 4). If the secondary header search state signal NS_Hunting is at a logic state "0", the value 0x00 is output from the output terminal of the multiplexer 714. On the other hand, if the secondary header search state signal NS_Hunting is at a logic state "1", the value 0x03 is output from the output terminal of the multiplexer 714.

[0063] A first input terminal of an OR gate 708 within the byte pointer 404 receives the secondary header search state signal NS_Hunting. A second input terminal of the OR gate 708 receives a pointer signal Ptr_Max from the decoder 712. The output terminal of the OR gate 708 outputs an ORed signal OR2. A first input terminal of a NAND gate receives the data stream enable signal AtmRx_Hit. A second input terminal of the OR gate 708 receives the mask signal from the D-type flip-flop 704. The output terminal of the AND gate outputs an ANDed signal to the ENA terminal of a counter 710.

[0064] An input terminal D of the counter 710 within the byte pointer 404 receives the 6-bit multiplexed (MUX) signal from the multiplexer 714. The LOAD terminal of the counter 710 receives the ORed signal OR2 from the OR gate 708 and the clock CLK terminal of the counter 710 receives a synchronizing pulse. The ENA terminal of the counter 710 receives the signal AND from the AND gate 706. The output terminal Q of the counter 710 outputs a 6-bit count value Byte_Ptr. During a cycle of synchronizing pulse applied to the CLK terminal, if the OR gate signal OR2 and the AND gate signal are both in a logic state "1", the output terminal Q outputs a multiplex signal MUX. The four highest effective bits of data from the count value Byte_Ptr serves as a signal

PtrBuf_WrPtr. The signal PtrBuf_WrPtr is sent to the buffer 304 (refer to Fig. 3) serving as an address for the data cell.

- [0065] After receiving the position pointer Byte_Ptr, the decoder 712 within the byte pointer 404 decodes the position pointer Byte_Ptr into several pointer signals and provides logic decision support to various modules in the transmission convergence sublayer. The states of these pointer signals are shown in Fig. 7A.
- [0066] In Fig. 4, the header cyclic redundancy checker 410 is coupled to the cell delineation state machine 402 for receiving the header within the consecutive data cells (as shown in Fig. 1) and computing out the 5 bytes of header data using the polynomial $X^8 + X^2 + X + 1$. Hence, a syndrome code is obtained. According to whether the syndrome code has a value 0x00 or not, correctness of the header within the data cells in a data transmission can be determined.
- [0067] Fig. 8A is a diagram showing a circuit implementation of the header cyclic redundancy checker according to one preferred embodiment of this invention. As shown in Fig. 8A, a remainder compensation unit 812 within the header cyclic redundancy checker 410 receives the bytes Pipe5 submitted by the byte-wise data pipeline 408 (refer to Fig. 4). The remainder compensation unit 812 conducts a XOR computation of the bytes Pipe5 to obtain remainder compensation data. Since bytes are temporarily stored in the byte-wise data pipeline 408, conventional flip-flop registers like the one shown in Fig. 2B is not required.
- [0068] A modulo 2 adder 814 within the header cyclic redundancy checker 410 adds together the header bytes provided by the data stream AtmRx_Data from the deframer 306 (refer to Fig. 3) and the remainder compensation data to form byte data ADD1. A second modulo 2 adder 816 adds together the byte data ADD1 and the quotient feedback data from a quotient feedback unit 810 to form byte data ADD2.
- [0069] The clock terminal CLK of a D-type flip-flop 806 receives a synchronizing pulse and the enable terminal ENA of the D-type flip-flop 806 receives the data stream enable signal AtmRx_Hit from the deframer 306 (refer to Fig. 3) so that the D-type flip-flop 806 is enabled. The output terminal Q of the D-type flip-flop 806 outputs flip-flop data FF_Output within a synchronizing pulse cycle.

[0070] The quotient feedback unit 810 within the header cyclic redundancy checker 410 conducts a XOR operation on the flip-flop data FF_Output from the D-type flip-flop 806 to produce the quotient feedback data. Inverters 818, 820, 822 and 824 conducts an inversion operation on the sixth, the fourth, the second and the zeroth bit of the flip-flop data FF_Output from the D-type flip-flop 806. After the inversion operation, the inverted bits and the non-inverted bits (the seventh, the fifth, the third and the first bit) of the flip-flop data FF_Output are combined together to form a syndrome code.

[0071] In Fig. 1, to ensure the received data cells are correct, the reception system introduces an 8-bit header cyclic redundancy code into the 5 byte header. The quotient feedback unit 810 as shown in Fig. 8A processes the header byte data. The quotient feedback unit 810 conducts a division operation using the polynomial $X^8 + X^2 + X + 1$ to produce the quotient feedback data. The quotient feedback data and the data stream AtmRx_Data as well as the remainder compensation data undergo modulo 2 additions by the modulo 2 adders 814 and 816 and the partially inverted by the inverters 818, 820, 822 and 824 to form the syndrome code. If the header of consecutively received data cells produces a syndrome code 0x00, correct consecutive header cells are implied and hence data cell reception may go ahead. On the contrary, if the syndrome code is different from 0x00, search for correct header is continued.

[0072] In Fig. 8A, the header within a data cell and the payload will be fed to the circuit in sequence. Only the syndrome code computed from the header is meaningful. The payload data is modified by the remainder compensation unit 812 through the acquisition of data Pipe5 from the byte-wise data pipeline 408.

[0073] Fig. 8B is a diagram showing a circuit implementation of the remainder compensation unit according to one preferred embodiment of this invention. As shown in Fig. 8B, a first input terminal of a XOR gate 831 within the remainder compensation unit 812 receives the seventh bit of byte data Pipe5, a second input terminal receives the sixth bit of byte data Pipe5, a third input terminal receives the second bit of the byte data Pipe5 and a fourth input terminal receives the first bit of the byte data pipe5. The XOR gate 831 conducts a XOR operation and outputs from its output terminal the seventh bit of the remainder compensation data.

[0075] A first input terminal of a XOR gate 833 receives the fifth bit of byte data Pipe5, a second input terminal receives the fourth bit of byte data Pipe5 and a third input terminal receives the zeroth bit of the byte data Pipe5. The XOR gate 833 conducts a XOR operation and outputs from its output terminal the fifth bit of the remainder compensation data.

[0076] A first input terminal of a XOR gate 834 receives the seventh bit of byte data Pipe5, a second input terminal receives the fourth bit of byte data Pipe5 and a third input terminal receives the third bit of the byte data Pipe5. The XOR gate 834 conducts a XOR operation and outputs from its output terminal the fourth bit of the remainder compensation data.

[0077] A first input terminal of a XOR gate 835 receives the sixth bit of byte data Pipe5, a second input terminal receives the third bit of byte data Pipe5 and a third input terminal receives the second bit of the byte data Pipe5. The XOR gate 835 conducts a XOR operation and outputs from its output terminal the third bit of the remainder compensation data.

[0078] A first input terminal of a XOR gate 836 receives the seventh bit of byte data Pipe5, a second input terminal receives the fifth bit of byte data Pipe5, a third input terminal receives the second bit of the byte data Pipe5 and a fourth input terminal receives the first bit of the byte data pipe5. The XOR gate 836 conducts a XOR operation and outputs from its output terminal the second bit of the remainder compensation data.

[0079] A first input terminal of a XOR gate 837 receives the seventh bit of byte data Pipe5, a second input terminal receives the fourth bit of byte data Pipe5, a third input terminal receives the second bit of the byte data Pipe5 and a fourth input terminal receives the zeroth bit of the byte data pipe5. The XOR gate 837 conducts a XOR

operation and outputs from its output terminal the first bit of the remainder compensation data.

[0080] A first input terminal of a XOR gate 838 receives the seventh bit of byte data Pipe5, a second input terminal receives the third bit of byte data Pipe5 and a third input terminal receives the second bit of the byte data Pipe5. The XOR gate 838 conducts a XOR operation and outputs from its output terminal the zeroth bit of the remainder compensation data.

[0081] Fig. 8C is a diagram showing a circuit implementation of the quotient feedback unit according to one preferred embodiment of this invention. As shown in Fig. 8C, a first input terminal of a XOR gate 841 within the quotient feedback unit 810 receives the seventh bit of the flip-flop data FF_Output, a second input terminal receives the sixth bit of the flip-flop data FF_Output and a third input terminal receives the fifth bit of the flip-flop data FF_Output. The XOR gate 841 conducts a XOR operation and outputs from its output terminal the seventh bit of the quotient feedback data.

[0082] A first input terminal of a XOR gate 842 receives the sixth bit of the flip-flop data FF_Output, a second input terminal receives the fifth bit of the flip-flop data FF_Output and a third input terminal receives the fourth bit of the flip-flop data FF_Output. The XOR gate 842 conducts a XOR operation and outputs from its output terminal the sixth bit of the quotient feedback data.

[0083] A first input terminal of a XOR gate 843 receives the fifth bit of the flip-flop data FF_Output, a second input terminal receives the fourth bit of the flip-flop data FF_Output and a third input terminal receives the third bit of the flip-flop data FF_Output. The XOR gate 843 conducts a XOR operation and outputs from its output terminal the fifth bit of the quotient feedback data.

[0084] A first input terminal of a XOR gate 844 receives the fourth bit of the flip-flop FF_Output, a second input terminal receives the third bit of the flip-flop data FF_Output and a third input terminal receives the second bit of the flip-flop data FF_Output. The XOR gate 844 conducts a XOR operation and outputs from its output terminal the fourth bit of the quotient feedback data.

[0085] A first input terminal of a XOR gate 845 receives the seventh bit of the flip-flop

data FF_Output, a second input terminal receives the third bit of the flip-flop data FF_Output, a third input terminal receives the second bit of the flip-flop data FF_Output and a fourth input terminal receives the first bit of the flip-flop data FF_Output. The XOR gate 845 conducts a XOR operation and outputs from its output terminal the third bit of the quotient feedback data.

[0086] A first input terminal of a XOR gate 846 receives the sixth bit of the flip-flop data FF_Output, a second input terminal receives the second bit of the flip-flop data FF_Output, a third input terminal receives the first bit of the flip-flop data FF_Output and a fourth input terminal receives the zeroth bit of the flip-flop data FF_Output. The XOR gate 846 conducts a XOR operation and outputs from its output terminal the second bit of the quotient feedback data.

[0087] A first input terminal of a XOR gate 847 receives the sixth bit of the flip-flop data FF_Output, a second input terminal receives the first bit of the flip-flop data FF_Output and a third input terminal receives the zeroth bit of the flip-flop data FF_Output. The XOR gate 847 conducts a XOR operation and outputs from its output terminal the first bit of the quotient feedback data.

[0088] A first input terminal of a XOR gate 848 receives the seventh bit of the flip-flop data FF_Output, a second input terminal receives the sixth bit of the flip-flop data FF_Output and a third input terminal receives the zeroth bit of the flip-flop data FF_Output. The XOR gate 848 conducts a XOR operation and outputs from its output terminal the zeroth bit of the quotient feedback data.

[0089] Fig. 8D is a diagram showing a circuit implementation of the header cyclic redundancy checker according to another preferred embodiment of this invention. As shown in Fig. 8D, a first input terminal of a XOR gate 856 within the header cyclic redundancy checker 410" receives the seventh bit of the byte Pipe4 from the byte-wise data pipeline 408 (Fig. 8D reference Fig. 4) and a second input terminal receives the first bit of the byte Pipe4 from the byte-wise data pipeline 408. The XOR gate 856 outputs from its output terminal a first XOR signal.

[0090] A first input terminal of a XOR gate 857 receives the sixth bit of the byte Pipe4 from the byte-wise data pipeline 408, a second input terminal receives the seventh bit

of the byte Pipe4 from the byte-wise data pipeline 408 and a third input terminal receives the zeroth bit of the byte Pipe4 from the byte-wise data pipeline 408. The XOR gate 857 outputs from its output terminal a second XOR signal.

[0091] A first input terminal of a XOR gate 858 receives the fifth bit of the byte Pipe4 from the byte-wise data pipeline 408, a second input terminal receives the sixth bit of the byte Pipe4 from the byte-wise data pipeline 408, a third input terminal receives the seventh bit of the byte Pipe4 from the byte-wise data pipeline 408 and a fourth input terminal receives the seventh bit of the byte Pipe3 from the byte-wise data pipeline. The XOR gate 858 outputs from its output terminal a third XOR signal.

[0092] A first input terminal of a XOR gate 859 receives the fourth bit of the byte Pipe4 from the byte-wise data pipeline 408, a second input terminal receives the fifth bit of the byte Pipe4 from the byte-wise data pipeline 408, a third input terminal receives the sixth bit of the byte Pipe4 from the byte-wise data pipeline 408 and a fourth input terminal receives the sixth bit of the byte Pipe3 from the byte-wise data pipeline. The XOR gate 859 outputs from its output terminal a fourth XOR signal.

[0093] A first input terminal of a XOR gate 860 receives the third bit of the byte Pipe4 from the byte-wise data pipeline 408, a second input terminal receives the fourth bit of the byte Pipe4 from the byte-wise data pipeline 408, a third input terminal receives the fifth bit of the byte Pipe4 from the byte-wise data pipeline 408 and a fourth input terminal receives the fifth bit of the byte Pipe3 from the byte-wise data pipeline. The XOR gate 860 outputs from its output terminal a fifth XOR signal.

[0094] A first input terminal of a XOR gate 861 receives the second bit of the byte Pipe4 from the byte-wise data pipeline 408, a second input terminal receives the third bit of the byte Pipe4 from the byte-wise data pipeline 408, a third input terminal receives the fourth bit of the byte Pipe4 from the byte-wise data pipeline 408 and a fourth input terminal receives the fourth bit of the byte Pipe3 from the byte-wise data pipeline. The XOR gate 861 outputs from its output terminal a sixth XOR signal.

[0095] A first input terminal of a XOR gate 862 receives the first XOR signal, a second input terminal receives the second bit of the byte Pipe4 from the byte-wise data pipeline 408, a third input terminal receives the third bit of the byte Pipe4 from the

byte-wise data pipeline 408 and a fourth input terminal receives the third bit of the byte Pipe3 from the byte-wise data pipeline 408. The XOR gate 862 outputs from its output terminal a seventh XOR signal.

[0096] A first input terminal of a XOR gate 863 receives the second XOR signal, a second input terminal receives the first XOR signal, a third input terminal receives the second bit of the byte Pipe4 from the byte-wise data pipeline 408 and a fourth input terminal receives the second bit of the byte Pipe3 from the byte-wise data pipeline 408. The XOR gate 863 outputs from its output terminal an eighth XOR signal.

[0097] A first input terminal of a XOR gate 864 receives the third XOR signal, a second input terminal receives the second XOR signal, a third input terminal receives the third XOR signal and a fourth input terminal receives the first bit of the byte Pipe3 from the byte-wise data pipeline 408. The XOR gate 864 outputs from its output terminal a ninth XOR signal.

[0098] A first input terminal of a XOR gate 865 receives the fourth XOR signal, a second input terminal receives the third XOR signal, a third input terminal receives the second XOR signal and a fourth input terminal receives the zeroth bit of the byte Pipe3 from the byte-wise data pipeline 408. The XOR gate 865 outputs from its output terminal a tenth XOR signal.

[0099] A first input terminal of a XOR gate 866 receives the fifth XOR signal, a second input terminal receives the fourth XOR signal, a third input terminal receives the third XOR signal and a fourth input terminal receives the seventh bit of the byte Pipe2 from the byte-wise data pipeline 408. The XOR gate 866 outputs from its output terminal an eleventh XOR signal.

[0100] A first input terminal of a XOR gate 867 receives the sixth XOR signal, a second input terminal receives the fifth XOR signal, a third input terminal receives the fourth XOR signal and a fourth input terminal receives the sixth bit of the byte Pipe2 from the byte-wise data pipeline 408. The XOR gate 867 outputs from its output terminal a twelfth XOR signal.

[0101] A first input terminal of a XOR gate 868 receives the seventh XOR signal, a second input terminal receives the sixth XOR signal, a third input terminal receives the fifth

XOR signal and a fourth input terminal receives the fifth bit of the byte Pipe2 from the byte-wise data pipeline 408. The XOR gate 868 outputs from its output terminal a thirteenth XOR signal.

[0102] A first input terminal of a XOR gate 869 receives the eighth XOR signal, a second input terminal receives the seventh XOR signal, a third input terminal receives the sixth XOR signal and a fourth input terminal receives the fourth bit of the byte Pipe2 from the byte-wise data pipeline 408. The XOR gate 869 outputs from its output terminal a fourteenth XOR signal.

[0103] A first input terminal of a XOR gate 870 receives the ninth XOR signal, a second input terminal receives the eighth XOR signal, a third input terminal receives the seventh XOR signal and a fourth input terminal receives the third bit of the byte Pipe2 from the byte-wise data pipeline 408. The XOR gate 870 outputs from its output terminal a fifteenth XOR signal.

[0104] A first input terminal of a XOR gate 871 receives the tenth XOR signal, a second input terminal receives the ninth XOR signal, a third input terminal receives the eighth XOR signal and a fourth input terminal receives the second bit of the byte Pipe2 from the byte-wise data pipeline 408. The XOR gate 871 outputs from its output terminal a sixteenth XOR signal.

[0105] A first input terminal of a XOR gate 872 receives the eleventh XOR signal, a second input terminal receives the tenth XOR signal, a third input terminal receives the ninth XOR signal and a fourth input terminal receives the first bit of the byte Pipe2 from the byte-wise data pipeline 408. The XOR gate 872 outputs from its output terminal a seventeenth XOR signal.

[0106] A first input terminal of a XOR gate 873 receives the twelfth XOR signal, a second input terminal receives the eleventh XOR signal, a third input terminal receives the tenth XOR signal and a fourth input terminal receives the zeroth bit of the byte Pipe2 from the byte-wise data pipeline 408. The XOR gate 873 outputs from its output terminal an eighteenth XOR signal.

[0107] A first input terminal of a XOR gate 874 receives the thirteenth XOR signal, a second input terminal receives the twelfth XOR signal, a third input terminal receives

the eleventh XOR signal and a fourth input terminal receives the seventh bit of the byte Pipe1 from the byte-wise data pipeline 408. The XOR gate 874 outputs from its output terminal a nineteenth XOR signal.

[0108] A first input terminal of a XOR gate 875 receives the fourteenth XOR signal, a second input terminal receives the thirteenth XOR signal, a third input terminal receives the twelfth XOR signal and a fourth input terminal receives the sixth bit of the byte Pipe1 from the byte-wise data pipeline 408. The XOR gate 875 outputs from its output terminal a twentieth XOR signal.

[0109] A first input terminal of a XOR gate 876 receives the fifteenth XOR signal, a second input terminal receives the fourteenth XOR signal, a third input terminal receives the thirteenth XOR signal and a fourth input terminal receives the fifth bit of the byte Pipe1 from the byte-wise data pipeline 408. The XOR gate 876 outputs from its output terminal a twenty-first XOR signal.

[0110] A first input terminal of a XOR gate 877 receives the sixteenth XOR signal, a second input terminal receives the fifteenth XOR signal, a third input terminal receives the fourteenth XOR signal and a fourth input terminal receives the fourth bit of the byte Pipe1 from the byte-wise data pipeline 408. The XOR gate 877 outputs from its output terminal a twenty-second XOR signal.

[0111] A first input terminal of a XOR gate 878 receives the seventeenth XOR signal, a second input terminal receives the sixteenth XOR signal, a third input terminal receives the fifteenth XOR signal and a fourth input terminal receives the third bit of the byte Pipe1 from the byte-wise data pipeline 408. The XOR gate 878 outputs from its output terminal a twenty-third XOR signal.

[0112] A first input terminal of a XOR gate 879 receives the eighteenth XOR signal, a second input terminal receives the seventeenth XOR signal, a third input terminal receives the sixteenth XOR signal and a fourth input terminal receives the second bit of the byte Pipe1 from the byte-wise data pipeline 408. The XOR gate 879 outputs from its output terminal a twenty-fourth XOR signal.

[0113] A first input terminal of a XOR gate 880 receives the nineteenth XOR signal, a second input terminal receives the eighteenth XOR signal, a third input terminal

receives the seventeenth XOR signal and a fourth input terminal receives the first bit of the byte Pipe1 from the byte-wise data pipeline 408. The XOR gate 880 outputs from its output terminal a twenty-fifth XOR signal.

[0114] A first input terminal of a XOR gate 881 receives the twentieth XOR signal, a second input terminal receives the nineteenth XOR signal, a third input terminal receives the eighteenth XOR signal and a fourth input terminal receives the zeroth bit of the byte Pipe1 from the byte-wise data pipeline 408. The XOR gate 881 outputs from its output terminal a twenty-sixth XOR signal.

[0115] A first input terminal of a XOR gate 882 receives the twenty-first XOR signal, a second input terminal receives the twentieth XOR signal, a third input terminal receives the nineteenth XOR signal and a fourth input terminal receives the seventh bit of the data stream AtmRx_Data. The XOR gate 882 outputs from its output terminal a twenty-seventh XOR signal to serve as the seventh bit of the syndrome code.

[0116] A first input terminal of a XOR gate 883 receives the twenty-second XOR signal, a second input terminal receives the twenty-first XOR signal, a third input terminal receives the twentieth XOR signal and a fourth input terminal receives the sixth bit of the data stream AtmRx_Data. The XOR gate 883 outputs from its output terminal a twenty-eighth XOR signal.

[0117] A first input terminal of a XOR gate 884 receives the twenty-third XOR signal, a second input terminal receives the twenty-second XOR signal, a third input terminal receives the twenty-first XOR signal and a fourth input terminal receives the fifth bit of the data stream AtmRx_Data. The XOR gate 884 outputs from its output terminal a twenty-ninth XOR signal to serve as the fifth bit of the syndrome code.

[0118] A first input terminal of a XOR gate 885 receives the twenty-fourth XOR signal, a second input terminal receives the twenty-third XOR signal, a third input terminal receives the twenty-second XOR signal and a fourth input terminal receives the fourth bit of the data stream AtmRx_Data. The XOR gate 885 outputs from its output terminal a thirtieth XOR signal.

[0119] A first input terminal of a XOR gate 886 receives the twenty-fifth XOR signal, a second input terminal receives the twenty-fourth XOR signal, a third input terminal

receives the twenty-third XOR signal and a fourth input terminal receives the third bit of the data stream AtmRx_Data. The XOR gate 886 outputs from its output terminal a thirty-first XOR signal to serve as the third bit of the syndrome code.

[0120] A first input terminal of a XOR gate 887 receives the twenty-sixth XOR signal, a second input terminal receives the twenty-fifth XOR signal, a third input terminal receives the twenty-fourth XOR signal and a fourth input terminal receives the second bit of the data stream AtmRx_Data. The XOR gate 887 outputs from its output terminal a thirty-second XOR signal.

[0121] A first input terminal of a XOR gate 889 receives the twenty-sixth XOR signal, a second input terminal receives the twenty-fifth XOR signal and a third input terminal receives the first bit of the data stream AtmRx_Data. The XOR gate 889 outputs from its output terminal a thirty-third XOR signal to serve as the first bit of the syndrome code.

[0122] A first input terminal of a XOR gate 890 receives the twenty-sixth XOR signal and a second input terminal receives the zeroth bit of the data stream AtmRx_Data. The XOR gate 890 outputs from its output terminal a thirty-fourth XOR signal.

[0123] An input terminal of an inverter 891 receives the twenty-eight XOR signal and outputs from its output terminal a first inverted signal to serve as the sixth bit of the syndrome code. An input terminal of an inverter 892 receives the thirtieth XOR signal and output from its output terminal a second inverted signal to serve as the fourth bit of the syndrome code. An input terminal of an inverter 893 receives the thirty-second XOR signal and output from its output terminal a third inverted signal to serve as the second bit of the syndrome code. An input terminal of an inverter 894 receives the thirty-fourth XOR signal and output from its output terminal a fourth inverted signal to serve as the zeroth bit of the syndrome code.

[0124] In brief, the byte data intercepted by the header cyclic redundancy checker in Fig. 8D is divided by the polynomial $X^8 + X^2 + X + 1$ to produce the syndrome code. In Fig. 8D, the five bytes within the data cell is accessed in parallel for computation. Under this arrangement, compensation of the payload portion of the data cell is unnecessary and the syndrome code thus obtained is identical to the one shown in Fig. 8A.

[0125] Fig. 8E is a diagram showing a circuit implementation of the header cyclic redundancy checker according to yet another preferred embodiment of this invention. In Fig. 8E, the header cyclic redundancy checker 410^{'''} differs from the one in Fig. 8A mainly in that an additional energy-saving device comprising a multiplexer 817 and an OR gate 815 is introduced. Input terminals of the OR gate 815 receives the pointer signals Ptr_L4 and Ptr_Max from the byte pointer 404 (refer to Fig. 4) and outputs an ORed signal to the multiplexer 817. The first input terminal "1" of the multiplexer 817 receives the byte data ADD2 from the modulo 2 adder 816^{''} and the second input terminal "0" of the multiplexer 817 receives the byte data 0x00. The output terminal of the multiplexer 817 outputs a multiplexed data byte SEL to the D-type flip-flop 806^{''}. The multiplexed data byte SEL is either the byte data ADD2 or the byte data 0x00 selected according to the OR signal from the OR gate 815. Since the rest of the components in Fig. 8E are identical to the ones in Fig. 8A, they are labeled identically and detailed descriptions of the components are not repeated here.

[0126] The purpose of incorporating the multiplexer 817 and the OR gate 815 is to save energy. When the transmission convergence sublayer 304 (refer to Fig. 4) is not in a searching state, or in other words, under pre-synchronization, full synchronization or synchronization conservation state, the header cyclic redundancy checker 410^{''} needs not detect whether each byte in a data cell is a header or not. The only operations required are the inspection after the last byte of each data cell (the pointer signal Ptr_Max) and the inspection of the foremost four bytes of each data cell (the pointer signal Ptr_L4).

[0127] In Fig. 4, the byte-wise data pipeline 408 receives the byte data within the data stream AtmRx_Data submitted by the deframer 306 (refer to Fig. 3). The data stream enable signal AtmRx_Hit is capable of enabling the byte-wise data pipeline 408. The byte-wise data pipeline 408 temporarily stores the byte data within the data stream AtmRx_Data according to the state signal from the cell delineation state machine 402 and the pointer signal from the byte pointer 404. The registered data stream AtmRx_Data data can be transmitted to the header cyclic redundancy checker 410 for compensatory modification in the decision for the reception of headers or to the header cyclic redundancy checker 410 for compensatory modification in the decision for the reception of headers. The data stream AtmRx_Data data may also send out two

double words of data in parallel (that is, 64 bits of data) to the descrambler 412 so that the descrambler 412 can descramble a double word of data to the buffer 304.

[0128] Fig. 9 is a diagram showing a circuit implementation of the byte-wise data pipeline according to one preferred embodiment of this invention. As shown in Fig. 9, an input terminal of an OR gate 922 within the byte-wise data pipeline 408 receives the state signal NS_Hunting from the cell delineation state machine 402 (refer to Fig. 4). An inversion input terminal of the OR gate 922 receives the pointer signal Ptr_03 from the byte pointer 404. The output terminal of the OR gate 922 outputs an ORed signal OR1. A first input terminal of the AND gate 920 receives the ORed signal OR1 from the OR gate 922 and a second input terminal of the AND gate 920 receives the data stream enable signal AtmRx_Hit. The output terminal of the AND gate 920 outputs a first enable signal Pipe_Enal to the enable terminal ENA of a first D-type flip-flop 902, a second D-type flip-flop 904, a third D-type flip-flop 906 and a fourth D-type flip-flop 908.

[0129] The clock terminal CLK of the D-type flip-flop 902 receives a synchronizing pulse while the enable terminal ENA of the D-type flip-flop 902 receives the first enable signal Pipe_Enal so that the D-type flip-flop 902 is enabled. The input terminal D of the D-type flip-flop 902 receives the data stream AtmRx_Data data and the output terminal Q of the D-type flip-flop 902 outputs the data byte Pipe1.

[0130] The clock terminal CLK of the D-type flip-flop 904 receives a synchronizing pulse while the enable terminal ENA of the D-type flip-flop 902 receives the first enable signal Pipe_Enal so that the D-type flip-flop 904 is enabled. The input terminal D of the D-type flip-flop 904 receives the data byte Pipe1 and the output terminal Q of the D-type flip-flop 904 outputs the data byte Pipe2.

[0131] The clock terminal CLK of the D-type flip-flop 906 receives a synchronizing pulse while the enable terminal ENA of the D-type flip-flop 906 receives the first enable signal Pipe_Enal so that the D-type flip-flop 906 is enabled. The input terminal D of the D-type flip-flop 906 receives the data byte Pipe2 and the output terminal Q of the D-type flip-flop 906 outputs the data byte Pipe3.

[0132] The clock terminal CLK of the D-type flip-flop 908 receives a synchronizing pulse

while the enable terminal ENA of the D-type flip-flop 908 receives the first enable signal Pipe_Enal so that the D-type flip-flop 908 is enabled. The input terminal D of the D-type flip-flop 908 receives the data byte Pipe3 and the output terminal Q of the D-type flip-flop 908 outputs the data byte Pipe4.

[0133] According to the pointer signal Ptr_03 and the state signal NS_Hunting, the data byte Pipe1, data byte Pipe2, data byte Pipe3 and data byte Pipe4 transmits serially from one D-type flip-flop to the next D-type flip-flop. However, the data bytes Pipe1, Pipe2, Pipe3 and Pipe4 may transmit in parallel a double word data (32 bits of data) to the descrambler 412. The pointer signal Ptr_03 controls the D-type flip-flop 902 such that the D-type flip-flop 902 is disabled after receiving the fourth byte H3. Hence, the fifth byte containing the header cyclic redundancy code HEC will not be received. The elimination of the header cyclic redundancy code prevents the incorporation of such code into the descrambler and the subsequent initiation of unnecessary descrambling.

[0134] An input terminal of an OR gate 928 receives the state signal NS_Hunting from the cell delineation state machine 402 and an inversion input terminal of the OR gate 928 receives the pointer signal Ptr_3t6 from the byte pointer 404. The output terminal of the OR gate 928 outputs an ORed signal OR2. A first input terminal of an AND gate 930 receives the ORed signal OR2 from the OR gate 928 and a second input terminal of the AND gate 930 receives the data stream enable signal AtmRx_Hit. The output terminal of the AND gate 930 outputs a second enable signal Pipe_Ena2 to a D-type flip-flop 934.

[0135] The clock terminal CLK of the D-type flip-flop 934 receives a synchronizing signal while the enable terminal ENA of the D-type flip-flop 934 receives the second enable signal Pipe_Ena2 from the AND gate 930. The input terminal D of the D-type flip-flop 934 receives the output from the D-type flip-flop 908. The output terminal Q of the D-type flip-flop 934 outputs not only to another flip-flop 910 but also to the header cyclic redundancy checker 410 via the pipeline Pipe5 so that payload data received by the byte-wise data pipeline 408 may be transmitted to the checker 410 for compensatory modification. Furthermore, through the disabling of the second enable signal Pipe_Ena2 (the pointer Ptr_3t6) during the acquisition of the fourth to the seventh bytes of a data cell from the byte-wise data pipeline 408, the header data H0

~ H3 acquired from the byte-wise pipeline 408 is shunt from the downstream flip-flops. Therefore, in subsequent descrambling, the descrambler 412 descrambles the payload data within the data cell. In addition, after receiving the eight bytes from the byte-wise data pipeline 408, the flip-flop 934 is disabled so that the payload portion of the data can be transferred to other flip-flops and then re-directed to the checker 410 for necessary compensation.

[0136] An inversion input terminal of an AND gate 932 within the byte-wise data pipeline 408 receives the pointer signal Ptr_3t6 from the byte pointer 404 and an input terminal of the AND gate 932 receives the data stream enable signal AtmRx_Hit. The output terminal of the AND gate 932 outputs a third enable signal Pipe_Ena3 to D-type flip-flops 910, 912, 914, 916 and 918 respectively.

[0137] The clock terminal CLK of the D-type flip-flop 910 receives a synchronizing signal while the enable terminal ENA of the D-type flip-flop 910 receives the third enable signal Pipe_Ena3 so that the D-type flip-flop 910 is enabled. The input terminal D of the D-type flip-flop 910 receives the output from the D-type flip-flop 934 and the output terminal Q of the D-type flip-flop 910 outputs a data byte Pipe6 and transmits the data byte Pipe6 to the D-type flip-flop 912 as well.

[0138] The clock terminal CLK of the D-type flip-flop 912 receives a synchronizing signal while the enable terminal ENA of the D-type flip-flop 912 receives the third enable signal Pipe_Ena3 so that the D-type flip-flop 912 is enabled. The input terminal D of the D-type flip-flop 912 receives the output from the D-type flip-flop 910 and the output terminal Q of the D-type flip-flop 912 outputs a data byte Pipe7 and transmits the data byte Pipe7 to the D-type flip-flop 914 as well.

[0139] The clock terminal CLK of the D-type flip-flop 914 receives a synchronizing signal while the enable terminal ENA of the D-type flip-flop 914 receives the third enable signal Pipe_Ena3 so that the D-type flip-flop 914 is enabled. The input terminal D of the D-type flip-flop 914 receives the output from the D-type flip-flop 912 and the output terminal Q of the D-type flip-flop 914 outputs a data byte Pipe8 and transmits the data byte Pipe8 to the D-type flip-flop 916 as well.

[0140] The clock terminal CLK of the D-type flip-flop 916 receives a synchronizing signal

while the enable terminal ENA of the D-type flip-flop 916 receives the third enable signal Pipe_Ena3 so that the D-type flip-flop 916 is enabled. The input terminal D of the D-type flip-flop 916 receives the output from the D-type flip-flop 914 and the output terminal Q of the D-type flip-flop 916 outputs a data byte Pipe9 and transmits the data byte Pipe9 to the D-type flip-flop 918 as well.

[0141] The clock terminal CLK of the D-type flip-flop 918 receives a synchronizing signal while the enable terminal ENA of the D-type flip-flop 918 receives the third enable signal Pipe_Ena3 so that the D-type flip-flop 918 is enabled. The input terminal D of the D-type flip-flop 918 receives the zeroth to the second bit of data from the D-type flip-flop 916 and the output terminal Q of the D-type flip-flop 918 outputs a 3-bit data byte Pipe10.

[0142] The consecutively received data cells are computed to obtain syndrome codes. If a syndrome code is 0x00, the payload data (refer to Fig. 1) within the data cell must be descrambled. The third to the seventh bit of data byte Pipe6, the data type Pipe7, the data byte Pipe8, the data byte Pipe9 and the 3-bit data byte Pipe10 are sent to the descrambler 412 as reference data for descrambling the payload data.

[0143] According to the circuit diagram in Fig. 9, the byte-wise data pipeline 408 not only receives and stores byte data, but also enables each section separately so that the pipeline 408 also serves as a common data source for supplying information to various related components. In other words, the flip-flops 902, 904, 906 and 908 controlled by the first enable signal Pipe_Enal can be considered as a first section. Similarly, the flip-flop 934 controlled by the second enable signal Pipe_Ena2 can be considered as a second section and the flip-flops 910, 912, 914, 916 and 918 can be considered as a third section.

[0144] In a search state, the first, the second and the third section are all disabled when the header is not yet found. Once the header is found and the foremost four bytes are acquired, the first, the second and the third sections are disabled according to the signals Ptr03 and Ptr3t6 received from the byte pointer 404 so that the acquisition of the header cyclic redundancy code (HEC) is blocked. After the pulse for receiving the fifth byte of the data cell is through, the first section is re-enabled. The second and the third section continues to be disabled until the seventh bytes is incorporated into

the byte-wise data pipeline 408 so that the header data H0 ~ H3 are blocked. This prevents the transmission of such data to ensuing flip-flops but permits the sequential input of payload data into the byte-wise data pipeline 408. Eventually, all sections are re-enabled to permit the reception of all the data cells.

[0145] In Fig. 4, the descrambler 412 receives the 64-bit data (the data byte Pipe1, the data byte Pipe2, the data byte Pipe3, the data byte Pipe4, the third bit to the seventh bit of the data byte Pipe6, the data byte Pipe7, the data byte Pipe8, the data byte Pipe9 and the 3-bit data byte Pipe10) from the byte-wise data pipeline 408. to the pointer signals submitted from the byte pointer 404, the descrambler 412 conducts a XOR operation of the double word data (the data bytes Pipe1, Pipe2, Pipe3 and Pipe4) having a scrambling format to obtain a descrambled double word data RxBuf_WrData. The double word data RxBuf_WrData is submitted to the buffer 304 (refer to Fig. 3).

[0146] Fig. 10 is a diagram showing a circuit implementation of the descrambler according to one preferred embodiment of this invention. As shown in Fig. 10, the multiple byte input terminal "0" of a multiplexer 1002 within the descrambler 412 receives the third bit to the seventh bit of the data byte Pipe6, the data byte Pipe7, the data byte Pipe8, the data byte Pipe9 and the 3-bit data byte Pipe10 sent from the byte-wise data pipeline 408 (refer to Fig. 4). The multiple byte input terminal "1" of the multiplexer 1002 receives the value 0x00000000. The select terminal SEL of the multiplexer 1002 receives the pointer signal Ptr_03 from the byte pointer 404 (refer to Fig. 4). The pointer signal Ptr_03 controls the multiplexer 1002 so that the multiple byte output terminal of the multiplexer 1002 outputs a multiple byte data MUX that includes the third bit to the seventh bit of the data byte Pipe6, the data byte Pipe7, the data byte Pipe8, the data byte Pipe9 and the 3-bit data byte Pipe10 or the value 0x00000000.

[0147] A first multiple byte input terminal of a XOR gate 1004 receives the data byte Pipe1, the data byte Pipe2, the data byte Pipe3 and the data byte Pipe4. A second multiple byte input terminal of the XOR gate 1004 receives the multiple byte data MUX from the multiplexer 1002. The XOR gate 1004 conducts a XOR operation of the 32-bit data sent to the first and the second multiple byte input terminal of the XOR gate

1004 to produce a double word data RxBuf_WrData. The double word data RxBuf_WrData is transmitted to the buffer 304 (refer to Fig. 3).

[0148] Before a transmitting system transmits necessary data cells to a reception system, the transmitting system often transmits an idle data cell to the reception system for data transmission synchronization. In Fig. 4, as the idle cell identifier 414 receives the data stream enable signal AtmRx_Hit, the idle cell identifier 414 is enabled. Thereafter, the idle cell identifier 414 may transmit an idle data cell signal to the write-in buffer controller 416 to inform the write-in buffer controller 416 that the data cell is an idle cell.

[0149] Fig. 11A is a diagram showing a data format of the idle cell according to this invention. Data format of the idle data cell includes header bytes H0 ~ H2 each having a value 0x00, header byte H3 having a value 0x01 and the header cyclic redundancy code HEC byte having a value 0x52.

[0150] Fig. 11B is a diagram showing a circuit implementation of the idle cell identifier according to one preferred embodiment of this invention. As shown in Fig. 11B, a first input terminal of a XOR gate 1102 within the idle cell identifier 414 receives the zeroth bit of the data byte Pipe1 from the byte-wise data pipeline 408 (refer to Fig. 4). A second input terminal of the XOR gate 1102 receives the pointer signal Ptr_03 from the byte pointer 404. The output terminal of the XOR gate 1102 outputs a XORed signal XOR_S.

[0151] A first input terminal of an OR gate 1104 receives a non-idle data cell signal Data_Cell from a D-type flip-flop 1112. A second to an eighth input terminal of the OR gate 1104 receive the seventh bit to the first bit of the data byte Pipe1 from the byte-wise data pipeline 408 respectively. A ninth input terminal of the OR gate 1104 receives the XORed signal XOR_S. The output terminal of the OR gate 1104 outputs an ORed signal OR_S1.

[0152] A first input terminal of an OR gate 1106 receives a pointer signal Ptr_Max from the byte pointer 404 and a second input terminal of the OR gate 1106 receives a pointer signal Ptr_L4 from the byte pointer 404. The output terminal of the OR gate 1106 outputs an ORed signal OR_S2. An input terminal of an AND gate 1110 receives

the ORed signal OR_S1 from the OR gate 1104 and an inversion input terminal of the AND gate 1110 receives the pointer signal Ptr_Max from the byte pointer 404. The output terminal of the AND gate 1110 outputs an ANDed signal AND_S.

[0153] A first input terminal of an AND gate 1108 receives the ORed signal OR_S2 from the OR gate 1106 and a second input terminal of the AND gate 1108 receives the data stream enable signal AtmRx_Hit. The output terminal of the AND gate 1108 outputs an ANDed enable signal AND_Ena.

[0154]

The input terminal D of a D-type flip-flop 1112 receives the signal AND_S and the enable terminal ENA of the D-type flip-flop 1112 receives the enable signal AND_Ena from the AND gate 1108. While the clock terminal CLK of the D-type flip-flop 1112 receives a synchronizing pulse, the output terminal Q of the D-type flip-flop 1112 outputs the non-idle data cell signal Data_Cell and the inverted output terminal

\bar{Q}

of the D-type flip-flop 1112 outputs an idle data cell signal Idle_Cell.

[0155] In Fig. 11B, as the idle cell identifier 414 circuit receives the non-idle data cell format, various non-idle cell signals Data_Cell produced by various gates within the idle cell identifier 414 are submitted to the write-in buffer controller 416 via the D-

type flip-flops 1112.

[0156] In Fig. 4, the write-in buffer controller 416 receives the data stream enable signal *AtmRx_Hit* so that the write-in buffer controller 416 is enabled. According to the pointer signal from the byte pointer 404, the state signals from the cell delineation state machine 402, the write request signal *RxBuf_WrReq* from the buffer 304 (refer to Fig. 3) and the non-idle data cell signal *Data_Cell* from the idle cell identifier 414, the write-in buffer controller 416 decides if the data cell needs to be sent to the buffer 304 (refer to Fig. 3). If the data cell is not an idle cell, the write-in buffer controller 416 submits a write-in signal *RxBuf_WrHit* to the buffer 304 (refer to Fig. 3) and informs the buffer 304 to receive the double word data from the descrambler 412. However, if the buffer 304 is completely filled, a signal *WrReq* will be submitted by the buffer 304 to inform the write-in buffer controller 416. When the descrambler 412 needs to submit double word data to the buffer 304, the write-in buffer controller 416 will issue an overflow signal *RxBuf_Ovf* to an upper layer system.

[0157] Fig. 12 is a diagram showing a circuit implementation of the write-in buffer controller according to one preferred embodiment of this invention. As shown in Fig. 12, a first input terminal of an OR gate 1202 within the write-in buffer controller 416 receives the pointer signal *Ptr_03* from the byte pointer 404 (refer to Fig 4). A second input terminal of the OR gate 1202 receives the non-idle data cell signal *Data_Cell* from the idle cell identifier 414. The output terminal of the OR gate 1202 outputs an ORed signal *OR*.

[0158] A first input terminal of an AND gate 1204 receives the ORed signal *OR* from the Or gate 1202. A second input terminal of the AND gate 1204 receives the state signal *NS_Corsync* from the cell delineation state machine 402. A third input terminal of the AND gate 1204 receives the pointer signal *Ptr_R3* from the byte pointer 404. A fourth input terminal of the AND gate 1204 receives the data stream enable signal *AtmRx_Hit*. The output terminal of the AND gate 1204 outputs an ANDed signal *AND*.

[0159] A first input terminal of an AND gate 1206 receives the ANDed signal *AND* from the AND gate 1204 and a second input terminal of the AND gate 1206 receives the write-in request signal *RxBuf_WrReq* from the buffer (refer to Fig. 3). The output terminal of the AND gate 1206 outputs a write-in signal *RxBuf_WrHit* to the buffer

[0160] An input terminal of an AND gate 1208 receives the ANDed signal AND from the AND gate 1204 and an inversion input terminal of the AND gate 1208 receives the write-in request signal RxBuf_WrReq from the buffer 304 (refer to Fig. 3). The output terminal of the AND gate 1208 outputs an overflow signal RxBuf_Ovf to an upper layer system.

[0161] In Fig. 12, as the write-in buffer 304 (refer to Fig. 3) is completely filled, the write-in request signal RxBuf_WrReq from the buffer 304 has a value "0". If the descrambler 412 (refer to Fig. 4) needs to write double word data into the buffer 304, the write-in buffer controller 416 will issue an overflow signal RxBuf_Ovf having a value "1" to upper layer system and inform the upper layer buffer 304 about the condition.

[0162] Fig. 13 is a block diagram showing a transmission convergence sublayer system for receiving asynchronous transfer according to a second embodiment of this invention. As shown in Fig. 13, the asynchronous transfer mode transmission convergence sublayer circuit 1300 at the reception end of a transmission system is similar to the one shown in Fig. 4. One major difference for the circuit in Fig. 13 is the addition of a header bit error corrector 1304. Furthermore, the header bit error corrector 1304 has terminals for sending signals to a descrambler 1502.

[0163] The header cyclic redundancy checker 410 computes to find syndrome code from the header cells (refer to Fig. 1). If a one-bit data error is found, the header cyclic redundancy checker 410 submits a syndrome code for the occurrence of one-bit data error to the header bit error corrector 1304. Fig. 14 is a lookup reference table for modifying bit errors. The header bit error corrector 1304 has a header bit error correction function. For example, when the zeroth bit of the header cell in error, the 8-bit syndrome code sent by the header cyclic redundancy checker 410 is 0x07. The header bit error corrector 1304 obtains a 32-bit correction code 0x0000001 from the header bit error table in Fig. 14. The 32-bit correction code is transmitted to the descrambler 1502. If the received header cell is correct or contains two or more bit errors, the syndrome code computed by the header cyclic redundancy checker 410 sets the header bit error checker 1304 to default and the header bit error corrector

1304 submits a correction code 0x00000000 to the descrambler 1502. In other words, no modification is carried out.

[0164] Fig. 15 is a diagram showing a circuit implementation of the descrambler for connecting with the header bit error corrector according to one preferred embodiment of this invention. In Fig. 15, the data received via a multiplexer 1504 within the descrambler 1502 must be corrected. A first multiple byte input terminal of the multiplexer 1504 receives the third bit to the seventh bit of the data byte Pipe6, the data byte Pipe7, the data byte Pipe8, the data byte Pipe9 and the 3-bit data byte Pipe10. A second multiple byte input terminal of the multiplexer 1504 receives the 32-bit correction code from the header bit error corrector 1304. The select SEL terminal of the multiplexer 1504 receives the pointer signal Ptr_03 from the byte pointer 404 (refer to Fig. 13). The pointer signal Ptr_03 controls the multiple byte output terminal of the multiplexer 1504 and selects receives the third bit to the seventh bit of the data byte Pipe6, the data byte Pipe7, the data byte Pipe8, the data byte Pipe9 and the 3-bit data byte Pipe10 or the 32-bit correction code as output. Thus, when the header in the data cell has a one-bit data error, the header bit error corrector 1304 can correct the error.

[0165] In conclusion, one major aspect of this invention is the provision of an optimized asynchronous transfer mode transmission convergence sublayer circuit to synchronize data reception, inspect header cell, descramble and conduct data format rearrangement within a shorter processing interval.

[0166] A second aspect of this invention is the ease of introducing a simple correction circuit to correct one-bit error in data cells during transmission. Thus, the circuit has error-correction capacity without adding too much complexity and cost to the fabrication of the circuit.

[0167] It will be apparent to those skilled in the art that various modifications and variations can be made to the structure of the present invention without departing from the scope or spirit of the invention. In view of the foregoing, it is intended that the present invention cover modifications and variations of this invention provided they fall within the scope of the following claims and their equivalents.